

An abstract graphic in the top-left corner consisting of a cluster of overlapping squares and circles in various shades of blue and white, creating a sense of depth and movement.

Python



Сферы применения Python

В порядке убывания охвата области и популярности языка в ней:

1. WEB (Django, Flask, aiohttp)
2. Data mining/нейросети (SciPy, NumPy)
3. Тестирование (PyTest)
4. Автоматизация (скрипты)
5. Системные утилиты (sys)
6. Desktop-приложения (PyQT)
7. Мобильные приложения (Kivy)



История языка

Язык программирования Python начал свою историю ещё в 1980-х годах, когда идеей о его создании загорелся Гвидо ван Россум - нидерландский программист. В декабре 1989 года он приступил к написанию языка Python в центре математики и информатики в Нидерландах. К 1991 была готова 1 версия интерпретатора.



Особенности языка

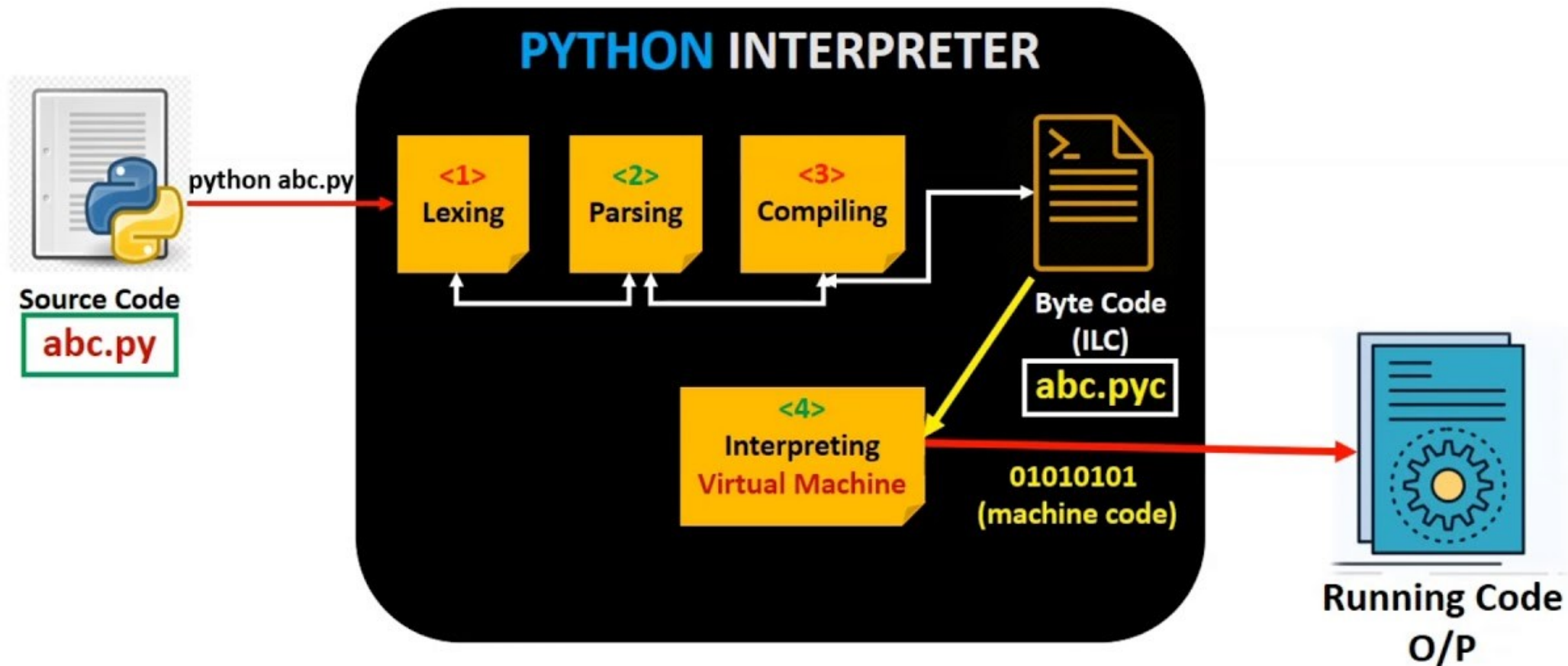
- Интерпретируемый язык высокого уровня
- Динамическая типизация
- Автоматический сборщик мусора
- Поддержка различных парадигм программирования
включая объектно-ориентированный подход




Как работает Python ?

1. Программа читается парсером и происходит анализ лексики. Где parser - это анализатор синтаксиса. В итоге получается набор лексем для дальнейшей обработки.
2. Затем парсером из инструкций происходит генерация структуры и формирования дерева синтаксического разбора- AST (Abstract Syntax Tree).
3. После этого компилятор преобразует AST в байт-код и отдает его на выполнение интерпретатору.

Simulating Python Interpreter

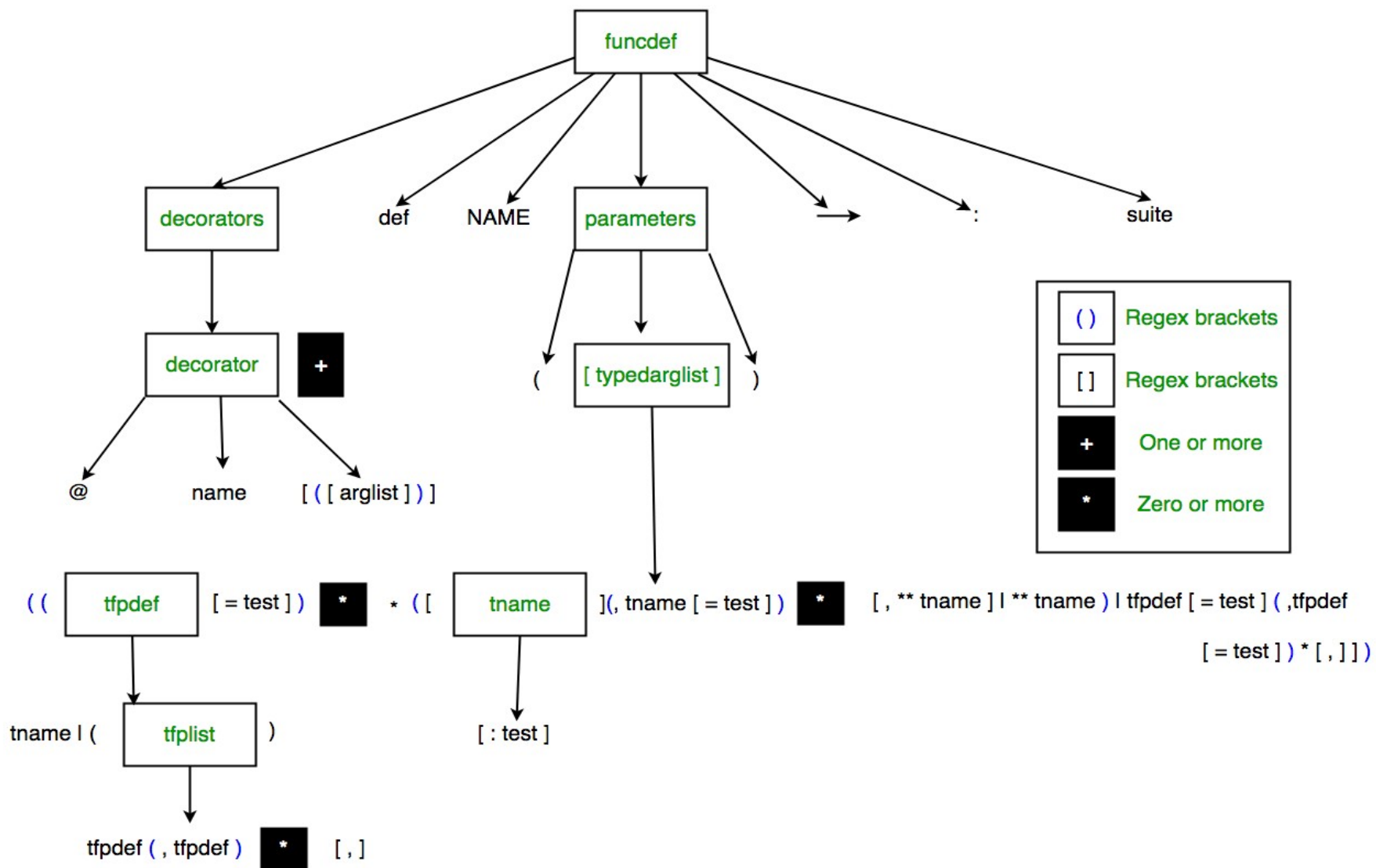




Что будет при интерпретации данного
кода ?

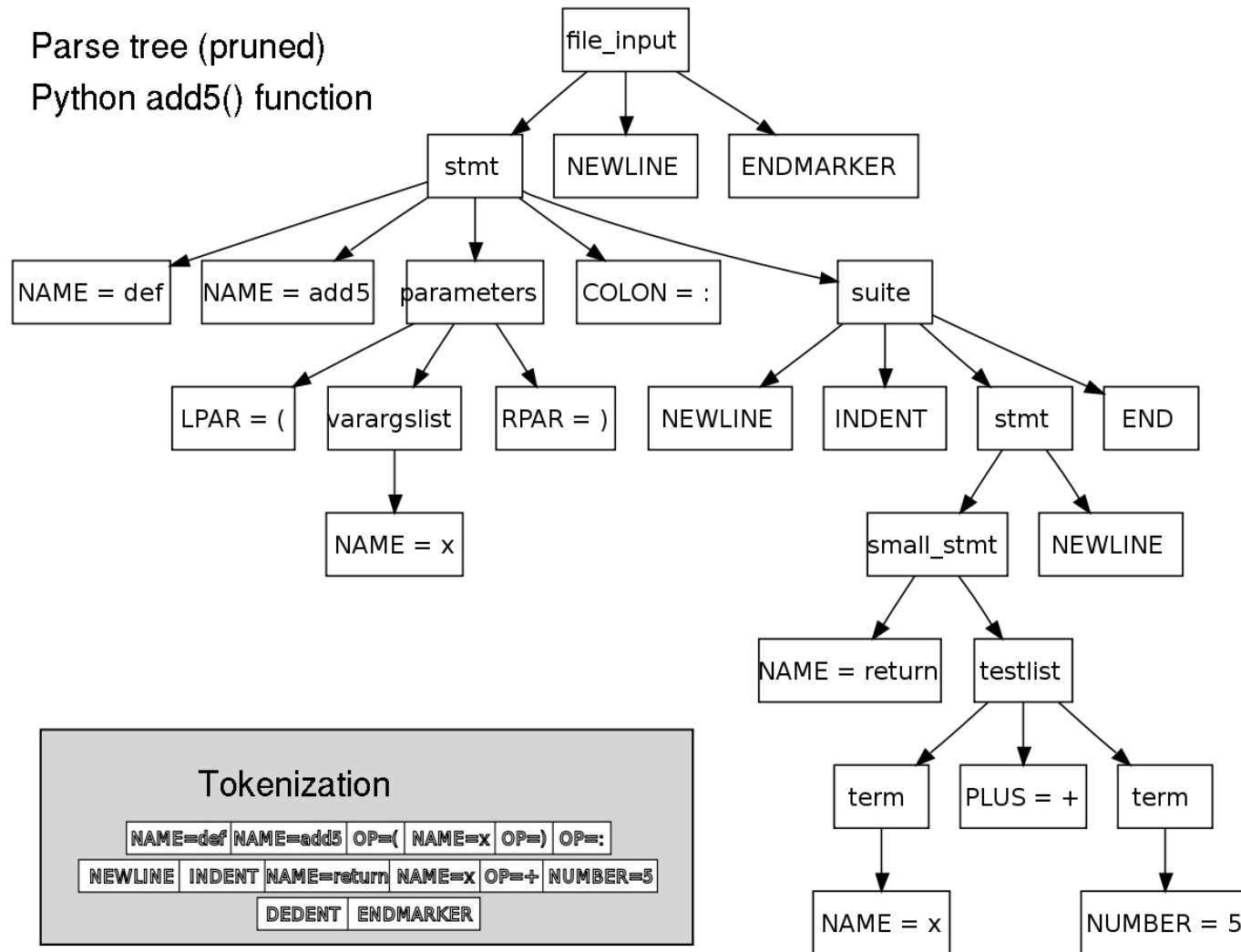
```
def summ5 (x) :  
    return x + 5
```

Грамматика функции




Дерево синтаксического разбора

Parse tree (pruned)
Python add5() function



Внутренние структуры хранения AST дерева

```
▼ Module: {} 2 keys
  ▼ body: [] 1 item
    ▼ 0: {} 1 key
      ▼ FunctionDef: {} 6 keys
        ► args: {} 1 key
        ► body: [] 1 item
        decorator_list: [] 0 items
        name: "summ5"
        returns: null
        type_comment: null
        type_ignores: [] 0 items
```



На стадии компиляции наш код превращается в байт код. В нашем случае бай код представлен мнемоническими именами. Затем он выполняется на виртуальной машине.

2	0 LOAD_FAST	0 (x)
	2 LOAD_CONST	1 (5)
	4 BINARY_ADD	
	6 RETURN_VALUE	



Для чего нам нужно знать про синтаксический разбор?

Если мы допускаем ошибки в грамматике кода то получаем синтаксическую ошибку.

SyntaxError: invalid syntax



Как писать без ошибок ?

Без ошибок писать пока не получится. Придется их устранять по ходу написания кода.

Чтобы писать без ошибок нужно следовать правилам формальной грамматики языка.



Формальные языки

Любой формальный язык, в том числе и Python, имеет три самые важные составляющие:

- * Операторы
- * Данные
- * Конструкции

Также в языках программирования часто присутствуют комментарии



Рассмотрим как пример один из самых известных формальных языков

$$(5 * 3 / (1+2))$$

В данном случае операторами являются

- * Оператор умножения
- * Оператор деления
- * Оператор сложения
- * Операторы группировки (скобочки)

Данными являются


- * Число 5
- * Число 3
- * Число 1
- * Число 2



Основы синтаксиса Python

Программа - это заданная последовательность инструкций. Инструкции выполняются сверху вниз.

- * Конец строки является концом инструкции (точка с запятой не требуется).
- * Вложенные инструкции объединяются в блоки по величине отступов. Отступ может быть любым. Отступ одинаков в пределах вложенного блока. В Python принят отступ в 4 пробела.

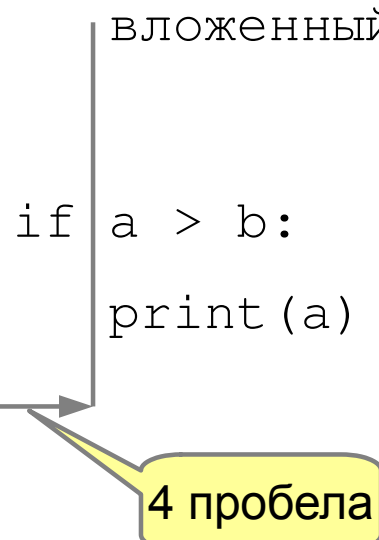


Некоторые операторы языка (if, for, try и т.д.) требуют вложенные инструкции. Они в Python записываются в соответствии с одним и тем же шаблоном. Когда основная инструкция завершается двоеточием, за ней идет вложенный блок кода с отступом.

основная инструкция:

вложенный блок

```
if a > b:  
    print(a)
```



4 пробела




Несколько случайных случаев

Иногда возможно записать несколько инструкций в одной строке, разделяя их точкой с запятой:


```
a = 1; b = 2; print(a, b)
```

Но не делайте это слишком часто! Помните об удобочитаемости. А лучше вообще так не делайте.



Допустимо записывать одну инструкцию в нескольких строках. Достаточно ее заключить в пару круглых, квадратных или фигурных скобок:

```
if (a == 1 and b == 2 and  
    c == 3 and d == 4): # Не забываем про двоеточие  
    print('spam' * 3)
```



Тело составной инструкции может располагаться в той же строке, что и тело основной, если тело составной инструкции не содержит составных инструкций. Пример:

```
if x > y: print(x)
```



Дальнейшее знакомство продолжим
на
практике программирования.

Разберем понятия переменной и типа
переменной.



Задача 1.

Определить в коде переменные:

1. Целочисленного типа
2. Вещественного типа
3. Логического типа
4. Строкового типа
5. Пустого типа

Вывести их типы.



Задача 2.

Преобразуйте переменную `age` и `foo` в число

```
age = "23"
```

```
foo = "23abc"
```

Преобразуйте переменную `age` в Boolean

```
age = 123abc
```

Преобразуйте переменную `flag` в Boolean

```
flag = 1
```

Преобразуйте значение в Boolean

```
str_one = "Privet"
```

```
str_two = ""
```

Преобразуйте значение 0 и 1 в Boolean

Преобразуйте False в строку.



Задача 3.

Данные две переменные:

```
age = 36.6
```

```
temperature = 25
```

Нужно обменять значения переменных местами. В итоге age должен равняться 25 а temperature – 36.6: